Datenbanken

I can’t wait for the SQL

Leon Muscat, Andrin Benz

Inhaltsverzeichnis

1 Modulbeschreibung

1.1 Lernziele

Studierende erwerben die Kompetenzen: - Datenbanken effektiv zu entwerfen und zu verwalten. - Grundlegende Kenntnisse für eine zukünftige Karriere in der IT-Industrie oder in verwandten Bereichen.

1.2 Veranstaltungs-Struktur und Lehr-/Lerndesign

Folgende Themen werden behandelt: - Datenmodelle (ER-Modell, relationales Modell). - Relationale Algebra und Kalkül. - SQL-Anfragesprache. - Dateiorganisation und Indexierung. - Anfrageverarbeitung und -optimierung. - Normalisierung. - Transaktionsverwaltung und Nebenläufigkeitskontrolle.

1.3 Literatur

- A. Silberschatz, H. F. Korth und S. Sudarshan: Database System Concepts. McGraw-Hill.
- R. Ramakrishnan und J. Gehrke: Database Management Systems. McGraw-Hill.
- R. Elmasri und S. B. Navathe: Fundamentals of Database Systems. Pearson.

1.4 Prüfung

1.4.1 1. Prüfungs-Teilleistung (1/2)

- **Typ:** Schriftliche Arbeit (Digital)
- **Organisation:** dezentral
- **Zeitpunkt:** Vorlesungszeit
- **Durchführung:** Asynchron, Off Campus
- **Benotung:** Gruppenarbeit (Gruppennote), 20%

1.4.2 2. Prüfungs-Teilleistung (2/2)

- **Typ:** Analoge schriftliche Prüfung
- **Organisation:** zentral
- **Zeitpunkt:** Vorlesungsfreie Zeit
- **Durchführung:** Synchron, On Campus
- **Benotung:** Einzelarbeit (Individualnote), 80%, 120 Min.

Bemerkungen: Beide Prüfungsteile sind in deutscher Sprache.

1.4.3 Hilfsmittel

1. **Freie Hilfsmittelregelung:** Studierende können grundsätzlich frei wählen, welche Hilfsmittel sie verwenden möchten. Einschränkungen werden durch die zuständigen Dozierenden definiert.
2. **Closed Book:** Die Benutzung von Hilfsmitteln ist untersagt, mit Ausnahme von Taschenrechnermodellen der Texas Instruments TI-30-Serie und zweisprachige Wörterbücher ohne Handnotizen.

1.4.4 Prüfungs-Inhalt

- Umfasst alle im Kurs behandelten Themen.

1.4.5 Prüfungs-Literatur

- Vorlesungs- und Übungsfolien
- Übungen
- Referenzen
- Diskussionen während der Vorlesung und Übungsstunden

2 Eigenschaften von Datenbanken

2.1 Komponenten eines Datenbankensystems

- **Datenbank (DB)**: Sammlung von Daten
- **Datenbankmanagementsystem (DBMS)**: Standardsoftware zur Definition, Verwaltung, Verarbeitung und Auswertung von DB-Daten
- **Datenbanksystem(DBS)**: DBS = DB + DBMS

2.2 Primäre Aufgaben eines DBS

- Beschreibung
- Speicherung und Pflege
- Wiedergewinnung

2.3 Anforderung an ein DBS

Edgar F. Codd ist der Vordenker von relationellen Datenbanken. Dabei hat er eine Liste von 9 Anforderungen an DBS formuliert.

Integration

- Einheitliche Verwaltung aller von Anwendungen benötigten Daten
- Redundanzfreie Datenhaltung des gesamten Datenbestands

Operationen

- Operationen zur Speicherung, Recherche und Manipulation der Daten müssen vorhanden sein

Data Dictionary

- Ein Katalog erlaubt Zugriffe auf die Beschreibung der Daten

Benutzersichten

- Für unterschiedliche Anwendungen unterschiedliche Sicht auf den Bestand

Konsistenzüberwachung

- Das DBMS überwacht die Korrektheit der Daten bei Änderungen

Zugriffskontrolle

- Ausschluss unautorisierter Zugriffe

Transaktionen

- Zusammenfassung einer Folge von Änderungsoperationen zu einer Einheit, deren Effekt bei Erfolg permanent in DB gespeichert wird

Synchronisation

- Arbeiten mehrere Benutzer gleichzeitig mit der DB, dann vermeidet das DBS unbeabsichtigte gegenseitige Beeinflussungen

Datensicherung

- Nach Systemfehlern oder Medienfehlern wird die Wiederherstellung ermöglicht (im Gegensatz zu Datei-Backup Rekonstruktion des Zustands der letzten erfolgreichen Transaktionen)

3 Entwurf einer Datenbank

Das Ziel einer jeden DB ist Genauigkeit, Aktualität, Verständlichkeit, Einfachheit und Redundanzfreiheit.

Viele Fehler und Probleme eines DBS fallen bereits in der Modellierungsphase auf. So können sie frühzeitig behandelt werden.

Die Modellierung von Datenbanken findet auf 3 Ebenen statt:

- **Konzeptionell**: Beschreibt die Gesamtsicht aller Daten der DB unabhängig von den einzelnen Applikationen. Es erklärt die Objekttypen und Beziehungen, aber nicht die Speicherung. Damit ist das konzeptionelle Datenmodell unabhängig des DBMS.
- **Physisch**: Beschreibt die systemspezifische Realisierung der DB-Objekte, also die interne Speicherung und Organisation der Daten (auch internes Schema genannt). Dadurch ist das physische Datenmodell abhängig des DBMS.
- **Logisch**: Beschreibt die Architektur und dient als Brücke zwischen dem konzeptionellen- und physischen Datenmodell. Es ist immer noch unabhängig von dem DBMS, aber bietet ein abstraktes Schema für das physische Datenmodell. Sehr bekannt ist das Relationenmodell.

Außerdem kann zwischen Views unterschieden werden. Views sind eine Sicht auf eine Teilmenge ds logischen Schemas für eine bestimmte Benutzergruppe. Dies dient vor allem für Übersichtlichkeit und Datenschutz.

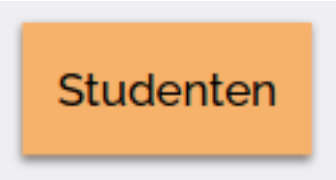
3.1 Konzeptionelles Design mit dem Entity-Relationship-Modell

Das Entity-Relationship-Modell (ER-Modell) modelliert Dinge (Entities) und deren Beziehungenen (Relationships). Es beantwortet also folgende Fragen:

- Welche Entitäten (= Objekte) gibt es in der Organisation?
- Welche Beziehungen existieren zwischen Entitäten?
- Welche Attribute (= Informationenspunkte) möchten wir über diese Entitäten und Beziehungen speichern?
- Welche Regeln gelten in der Organisation?
- Welche Integritätsbedingungen ergeben sich daraus?

Es existieren 3 Bausteine eines ER-Modells:

1. **Entities: Objekttypen**



Unterschieden wird zwischen Entitätstypen und Entitäten.

Entitätstypen sind wie Klassen, also Sammlungen ähnlicher Entitäten:

- Alle Entitäten eines Entitätstyps haben die gleiche Menge von Attributen
- Jedes Attribut hat einen Wertebereich (Domain)
- Jeder Entitätstype hat einen Schlüssel (eindeutige Identifikation)

Entitäten sind wie Instanzen von Entitätstypen, also unterscheidbare Objekte:

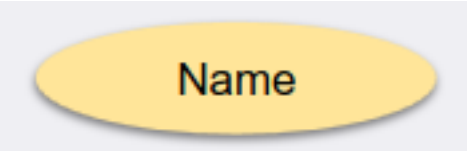
- Ausprägungen von Entitätstypen
- Eine Entität wird durch eine Reihe von Attributen beschrieben
- Entitäten tauchen nicht im ER-Diagramm

Wenn die Existenz einer Entität abhängig von einer anderen Entität ist, dann nennt man die abhängige Entität die untergeordnete oder schwache Entität und die andere Entität die dominierende oder starke Entität. Diese Beziehung nennt man Existenzbeziehung.

Die schwache Entität hat keinen eigenen Primärschlüssel, sondern erbt und erweitertert den Primärschlüssel der starken Entität.

Die schwache Entität und die Beziehung wird umkreist. Beispiel: Buchkapitel setzen die Existenz eines Buches voraus.

2. **Attribute: Eigenschaften**



Attribute sind Werte und beschreiben Eigenschaften von Entitäten und Entitätstypen.

Es gibt verschiedene Arten von Attributen:

- Einfache Attribute enthalten atomare Werte
- Zusammengesetzte Attribute kombinieren zwei oder mehrere Attribute
- Abgeleitete Attribute lassen sich einfachen Attributen errechnen (z.B. Summe von mehreren Attributen). Sie werden als gestrichelte Linie gekennzeichnet

- Mehrwertige Attribute lassen sich einer Entität mehrmals zuweisen (z.B. mehrere Telefonnummern) und werden doppelt umrandet
- Attribute, die den Schlüssel bilden, sind unterstrichen. Ein Schlüssel kann also durch Kombinationen von mehreren Attributen bestehen und beschreibt eine Entität eindeutig

3. **Relationships: Beziehungen zwischen Entities**

An einer Beziehung sind mindestens 2 Entitätstypen beteiligt. Es gibt One-to-One, One-to-Many und Many-to-Many Beziehungen.

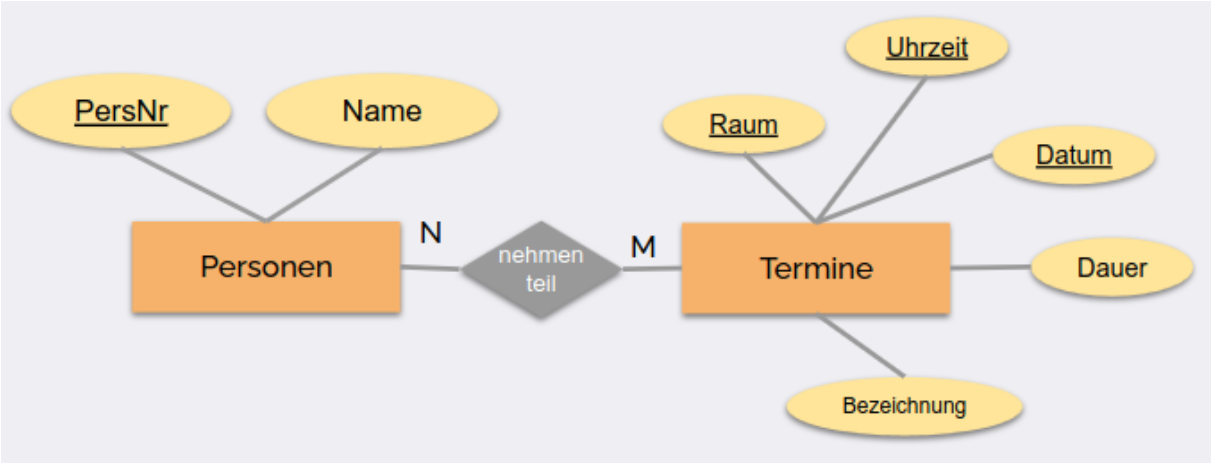
- One-to-One (1:1): Jede Person hat ein Gehirn
- One-to-Many (1:N): Jede Person kann N Termine haben und jeder Termin ist genau einer Person zugewiesen
- Many-to-Many (N:M): Jede Person kann N Termine haben und M Personen können einem Termin zugewiesen sein

Diese Unterscheidung wird Kardinalität einer Beziehung genannt.

Ein Entitätstyp kann auch rekursive Beziehungen haben. Das ist z.B. der Fall bei Ordner-Strukturen: Der Entitätstyp Ordner enthält Ordner oder Dateien und hat somit 2 Beziehungen wovon eine rekursiv ist.

Eine Beziehung kann binär sein bzw. Grad 2 haben, also nur 2 Entitäten verbinden. Sind z.B. 3 Entitäten an einer Beziehung beteiligt, dann wird diese ternär genannt und hat Grad 3.

Zusammen ergeben diese Komponenten ein ER-Modell:



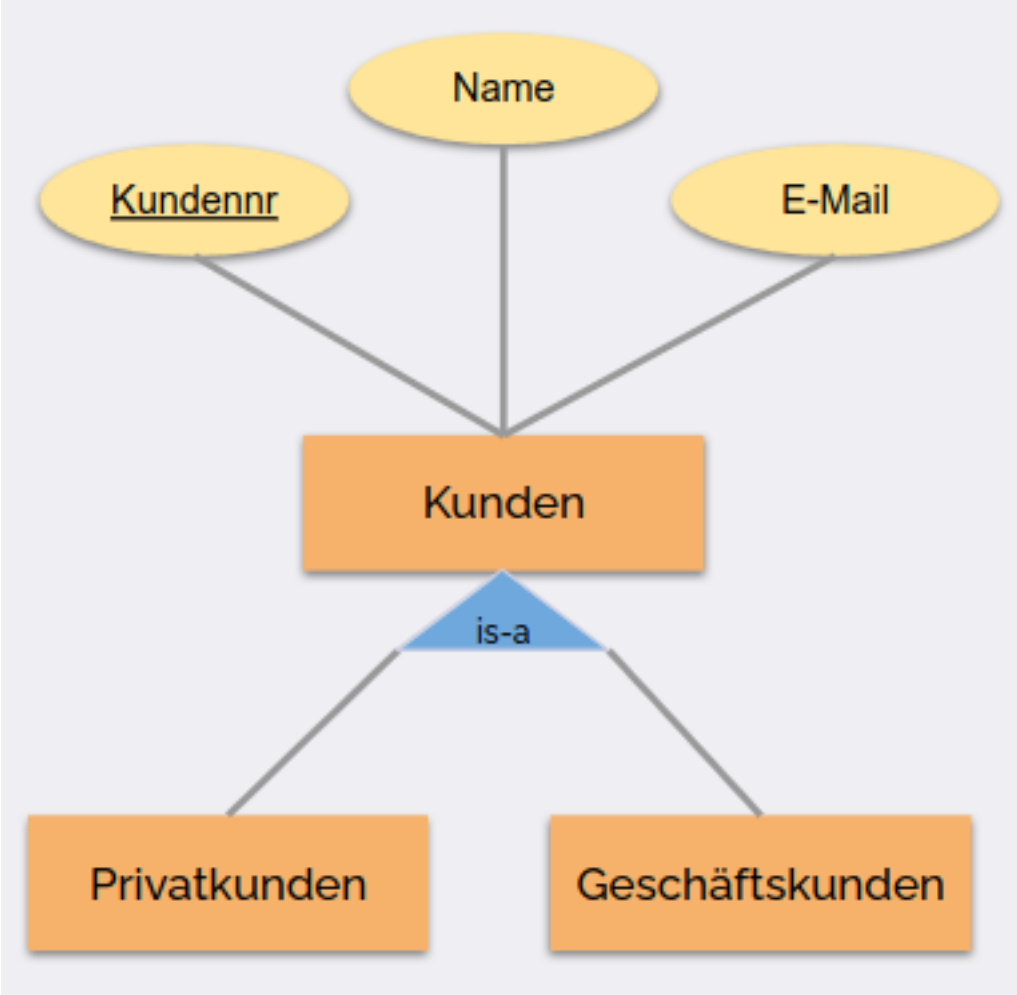
In der Regel gibt es mehrere Möglichkeiten ein realweltloches Konzept zu modellieren. Die Entscheidung auf eine Möglichkeit kann Auswirkungen auf die Redundanzen in den gespeicherten Daten und die Integritätsbedingungen, die von der Datenbankstruktur erfasst werden.

3.1.1 Generalisierung

So wie es in Java abstract classes gibt, gibt es in der ER-Modellierung Spezialisierung und Vererbung.

Dabei werden Attribute, Primärschlüssel und Beziehung des Super-Typen an die Sub-Typen vererbt.

Graphisch wird dies folgendermaßen dargestellt:



3.2 Integritätsbedingungen (Constraints)

Eien Integritätsbedingung ist eine Aussage über die Datenbank, die jederzeit erfüllt sein muss. Sie können genutzt werden, um den Zustand der Datenbanken zu überprüfen.

Sie spiegeln Fakten wider, die in der Welt existieren, bzw. Geschäftsregeln einer Organisation.

Beispiele:

- Schlüssel: Identifiziert eine Person eindeutig
- Einschränkung für Einzelwerte: Eine Person kann nur einen Vater haben
- Referentielle Integritätsbedingungen: Wenn eine Person für ein Unternehmen arbeitet, dann muss ein entsprechender Eintrag in der Datenbank vorhanden sein
- Domäneneinschränkung: Das Alter von Personen liegt zwischen 0 und 150 Jahren
- Kardinalitätsbedingungen: An einem Kurs können maximal 100 Studierende teilnehmen.

Integritätsbedingungen sind Teil des Datenbankschemas und sind somit Teil des Modellierungsprozesses.

3.3 Beispiel: Universitätsdatenbank

Im Folgenden gehen wir die Modellierung einer DB durch.

3.3.1 Was möchten wir wissen?

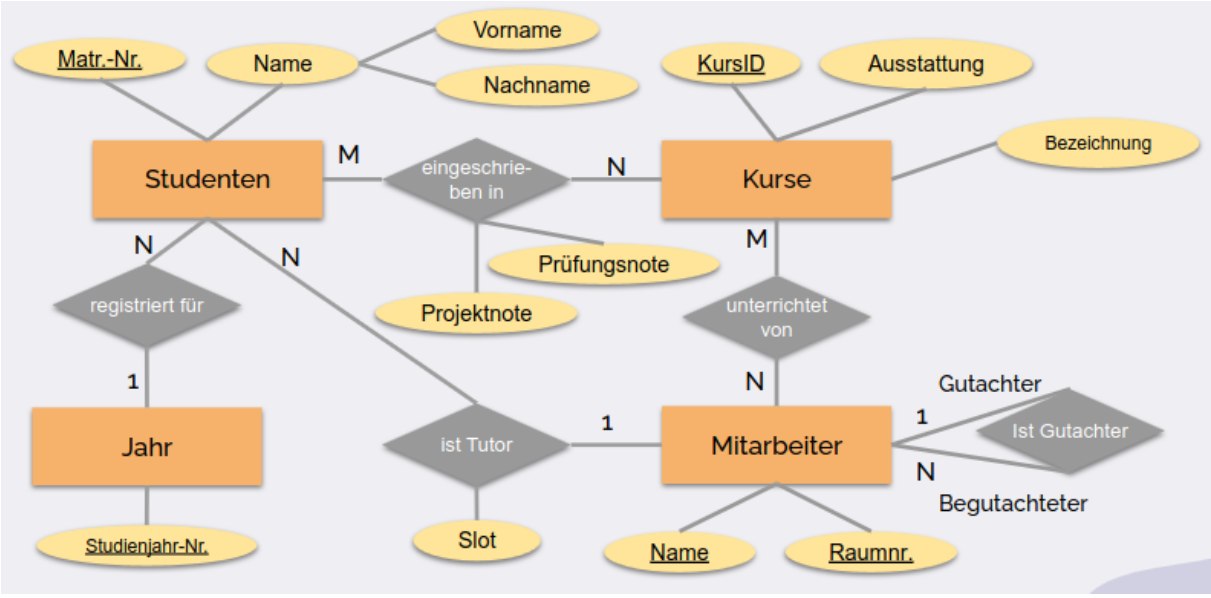
- Wie lauten der Vor- und Nachname des Studierenden mit der Matrikelnummer 54321
- Wie viele Studierende sind für den Kurs “Einführung in Datenbanken” eingeschrieben
- Für welche Kurse ist Georg Egger eingeschrieben
- Welche Kurse gibt Professor Schmidt und wann finden diese statt
- …

3.3.2 Was sind die Anforderungen?

- Ein Studierender hat einen Namen, der aus einem Vornamen und einem Nachnamen besteht, sowie eine Matrikelnummer. Jeder Studierende wird durch seine/ihre Matrikelnummer eindeutig identifiziert.
- Jedem Kurs ist eine Bezeichnung und eine Kurs-ID zugeordnet. Für jeden Kurs möchten wir die Anzahl der Studierenden erfassen, die den Kurs belegen, sowie die Art der Ausstattung, die für den Kurs verwendet wird. Ein Kurs wird durch seine Kurs-ID eindeutig identifiziert.

- Ein Studierender kann eine beliebige Anzahl von Kursen belegen und eine beliebige Anzahl Studierender kann in einem Kurs eingeschrieben sein. Für jeden Kurs, in dem sie eingeschrieben sind, erhalten die Studierenden eine Projekt- und eine Prüfungsnote.
- Ein Studierender ist auch für ein Studienjahr eingeschrieben. Ein Studienjahr wird durch eine Zahl zwischen 1 und 4 identifiziert. Ein Studierender ist nur für ein Studienjahr eingeschrieben, aber jede Kohorte kann beliebig viele Studierende haben.
- Für jedes Mitglied des Personals möchten wir deren Namen und Raumnummer erfassen. Ein Mitglied des Personals wird durch die Kombination dieser beiden Daten identifiziert. Das Personal wird von anderen Mitarbeitern bewertet. Ein Mitglied des Personals hat höchstens einen Gutachter.
- Kurse werden von Mitgliedern des Personals unterrichtet. Ein Kurs kann mehrere Dozierende haben und ein Mitarbeiter kann mehrere Kurse unterrichten.
- Studierende können einem Mitglied des Personals als Tutor zugeteilt werden. Ein Studierender kann höchstens einen Tutor haben. Der Tutor und der Studierende vereinbaren einen Zeitraum für regelmässige Treffen.

3.3.3 ER-Modell



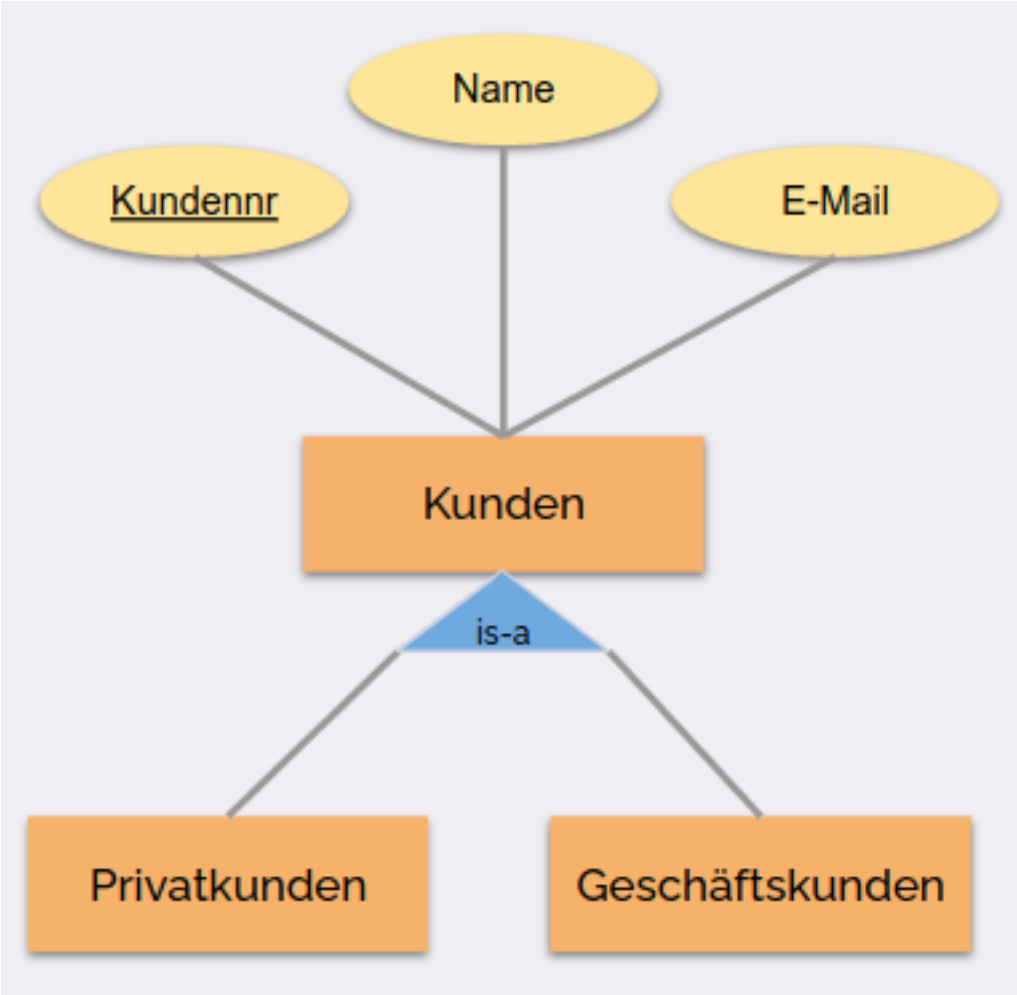
In der Regel gibt es mehrere Möglichkeiten ein realweltloches Konzept zu modellieren. Die Entscheidung auf eine Möglichkeit kann Auswirkungen auf die Redundanzen in den gespeicherten Daten und die Integritätsbedingungen, die von der Datenbankstruktur erfasst werden.

3.3.4 Generalisierung

So wie es in Java abstract classes gibt, gibt es in der ER-Modellierung Spezialisierung und Vererbung.

Dabei werden Attribute, Primärschlüssel und Beziehung des Super-Typen an die Sub-Typen vererbt.

Graphisch wird dies folgendermaßen dargestellt:



3.4 Integritätsbedingungen (Constraints)

Eien Integritätsbedingung ist eine Aussage über die Datenbank, die jederzeit erfüllt sein muss. Sie können genutzt werden, um den Zustand der Datenbanken zu überprüfen.

Sie spiegeln Fakten wider, die in der Welt existieren, bzw. Geschäftsregeln einer Organisation.

Beispiele:

- Schlüssel: Identifiziert eine Person eindeutig
- Einschränkung für Einzelwerte: Eine Person kann nur einen Vater haben
- Referentielle Integritätsbedingungen: Wenn eine Person für ein Unternehmen arbeitet, dann muss ein entsprechender Eintrag in der Datenbank vorhanden sein
- Domäneneinschränkung: Das Alter von Personen liegt zwischen 0 und 150 Jahren
- Kardinalitätsbedingungen: An einem Kurs können maximal 100 Studierende teilnehmen.

Integritätsbedingungen sind Teil des Datenbankschemas und sind somit Teil des Modellierungsprozesses.

3.5 Generelles zum ER-Diagramm

- Don't say the same thing more than once. (Vermeide Redundanz)
- Modelliere ein Konzept nur dann als Entität, wenn
  - es sich nicht nur um einen Namen für etwas handelt, d.h. Es über Nicht-Schlüsselattribute verfügt oder Beziehungen zu mehreren verschiedenen Entitäten aufweist
  - es das "Many" in einer Many-to-One Beziehung ist
- Checkliste: ER-Diagramm erstellen
  - Sind alle Kardinalitäten an Beziehungen?
  - Kardinalitäten richtig herum?
  - Gibt es überflüssige Entitätstypen?
  - Hat jeder Entitätstyp einen Primärschlüssel?
  - Müssen manche Entitätstypen schwach sein?
  - Ist die entsprechende Beziehung doppelt umrahmt?
  - Sind erweiternde Primärschlüssel "unterstrichelt"?
  - Sind Generalisierungen korrekt modelliert?

3.6 Beispiel: Universitätsdatenbank

Im Folgenden gehen wir die Modellierung einer DB durch.

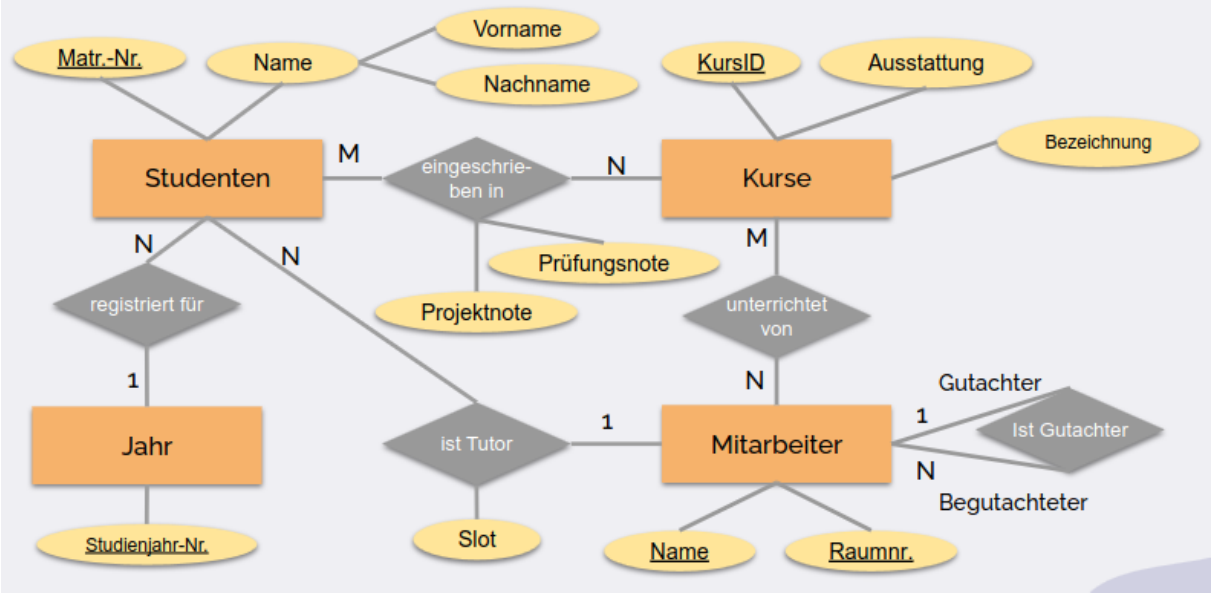
3.6.1 Was möchten wir wissen?

- Wie lauten der Vor- und Nachname des Studierenden mit der Matrikelnummer 54321
- Wie viele Studierende sind für den Kurs "Einführung in Datenbanken" eingeschrieben
- Für welche Kurse ist Georg Egger eingeschrieben
- Welche Kurse gibt Professor Schmidt und wann finden diese statt
- …

3.6.2 Was sind die Anforderungen?

- Ein Studierender hat einen Namen, der aus einem Vornamen und einem Nachnamen besteht, sowie eine Matrikelnummer. Jeder Studierende wird durch seine/ihre Matrikelnummer eindeutig identifiziert.
- Jedem Kurs ist eine Bezeichnung und eine Kurs-ID zugeordnet. Für jeden Kurs möchten wir die Anzahl der Studierenden erfassen, die den Kurs belegen, sowie die Art der Ausstattung, die für den Kurs verwendet wird. Ein Kurs wird durch seine Kurs-ID eindeutig identifiziert.
- Ein Studierender kann eine beliebige Anzahl von Kursen belegen und eine beliebige Anzahl Studierender kann in einem Kurs eingeschrieben sein. Für jeden Kurs, in dem sie eingeschrieben sind, erhalten die Studierenden eine Projekt- und eine Prüfungsnote.
- Ein Studierender ist auch für ein Studienjahr eingeschrieben. Ein Studienjahr wird durch eine Zahl zwischen 1 und 4 identifiziert. Ein Studierender ist nur für ein Studienjahr eingeschrieben, aber jede Kohorte kann beliebig viele Studierende haben.
- Für jedes Mitglied des Personals möchten wir deren Namen und Raumnummer erfassen. Ein Mitglied des Personals wird durch die Kombination dieser beiden Daten identifiziert. Das Personal wird von anderen Mitarbeitern bewertet. Ein Mitglied des Personals hat höchstens einen Gutachter.
- Kurse werden von Mitgliedern des Personals unterrichtet. Ein Kurs kann mehrere Dozierende haben und ein Mitarbeiter kann mehrere Kurse unterrichten.
- Studierende können einem Mitglied des Personals als Tutor zugeteilt werden. Ein Studierender kann höchstens einen Tutor haben. Der Tutor und der Studierende vereinbaren einen Zeitraum für regelmässige Treffen.

3.6.3 ER-Modell



4 Relationales Modell

4.1 Mathematische Grundlagen

4.1.1 Domain (Wertebereich)

- Eine Menge von Werten, die für ein Attribut zulässig sind
- Kann endliche oder unendliche Kardinalität haben

4.1.2 Kartesisches Produkt

- Das kartesische Produkt zweier Mengen A und B ist die Menge aller geordneten Paare (a, b), wobei $a \in A$ und $b \in B$
- Menge aller möglichen Kombinationen von Elementen aus A und B

4.1.3 Relationen

- es gibt endliche und unendliche Relationen (auf Datenbankebene nur endliche)
- Die einzelnen Domains lassen sich als Spalte einer Tabelle darstellen und werden als **Attribute** bezeichnet

- Relationen sind Mengen von Tupeln
  - Die Reihenfolge der Tupel ist irrelevant
  - Es gibt keine Duplikate

4.1.4 Schlüssel

Eine Teilmenge S der Attribute eines Relationenschemas R ist ein Schlüssel, wenn gilt: - Eindeutigkeit - Minimalität

4.1.4.1 Superschlüssel (Minimale Menge)

- Menge von Attributen (Spalten) in einer Relation (Tabelle), die die Tupel (Zeilen) eindeutig identifizieren.
- Ein trivialer Superschlüssel umfasst alle Attribute einer Relation, auch unnötige Spalten.

4.1.4.2 Schlüsselkanidat

- Minimale Teilmenge der Attribute eines Superschlüssels, die zur Identifizierung der Tupel benötigt werden.
- Kann nicht weiter reduziert werden, ohne die Eindeutigkeit zu verlieren.

4.1.5 Abhängigkeit

A und B sind Attributmengen aus der Relation R, dann sind A und B funktional abhängig, wenn der gleiche Wert für A immer den gleichen Wert B in der Zeile hat:

$$A \to B.$$

Gilt $\nexists X \subset A : X \to B$, dann sind A und B voll funktional abhängig:

$$A \implies B.$$

4.2 Transformation ER-Diagramm ins Relationenmodell

ER-Diagramm	Relationenmodell
Entitätstyp	Relation (=Tabelle)

ER-Diagramm	Relationenmodell
Attribut	Attribut (=Spalte)
Primärschlüssel	Primärschlüssel
Sub-Attribute	Einzelne Attribute
Mehrwertiges Attribut	Relation
1:N-Beziehung	Fremdschlüssel
N:M-Beziehung	Relation
Schwache Entitätstypen	Relation
Generalisierung	Relation(en)

4.2.1 Generalisierung

4.2.1.1 Volle Redundanz

- Jeder Entitätstyp wird zur eigenständigen Relation (alle Spalten)
- Beim Einfügen in Sub-Relationen wird redundante Information in die entsprechenden Super-Relationen eingefügt

4.2.1.2 Hausklassenmodell

- Jeder Entitätstyp wird zur eigenständigen Relation (alle Spalten)
- Es wird nur die speziellste Relation eingefügt

4.2.1.3 Vertikale Partitionierung

- Jeder Entitätstyp wird zur eigenständigen Relation (PK + spezielle Partitionierung)

4.2.1.4 Hierarchierelation

- Nur eine einzige Relation mit ALLEN Spalten Attribut Type_Tag gibt den Entitätstyp an

Kunden	KundenNr	Name	E-Mail	Bonuspunkte	USt-IdNr	Type_Tag
	4	Ute	ute@example.com	-	-	Kunde
	5	Peter	peter@example.com	2811	-	Privatkunde
	8	Anna	anna@example.com	-	555-12-3-456789	Geschäftskunde

5 Normalisierung

Die Normalisierung von Datenbanken ist ein zentraler Schritt im Designprozess, um Redundanzen zu minimieren und die Integrität der Daten zu sichern. Sie umfasst mehrere “Normalformen”, von denen jede darauf abzielt, bestimmte Arten von Anomalien und Redundanzen in den Daten zu beseitigen. Das erleichtert die Wartbarkeit der Datenbank, da Änderungen an einem Teil der Datenbanken ohne Auswirkungen auf andere Teile vorgenommen werden können.

Ein gut durchdachtes ER-Diagramm liegts bereits weitgehend normalisiert vor.

Normalisierung kann schädlich für die Performanz sein, weil Joins sehr teuer sind.

5.1 Erste Normalform (1NF)

Die erste Normalform (1NF) fordert, dass in einer Tabelle jedes Attribut atomare Werte enthält und jede Zeile eindeutig identifizierbar ist.

Beispiel einer nicht normalisierten Tabelle:

MatrikelNr	Name	Kurse
54321	Georg Egger	Datenbanken, Mathematik, Physik
12345	Anna Bauer	Informatik, Datenbanken

Probleme dieser Tabelle: - Die Kurse sind als eine zusammenhängende Zeichenkette statt als separate Einträge gespeichert. - Es gibt keine eindeutige Möglichkeit, einen bestimmten Kurs eines Studenten zu identifizieren.

Normalisierung in 1NF:

MatrikelNr	Name	Kurs
54321	Georg Egger	Datenbanken
54321	Georg Egger	Mathematik
54321	Georg Egger	Physik
12345	Anna Bauer	Informatik
12345	Anna Bauer	Datenbanken

5.2 Zweite Normalform (2NF)

Die zweite Normalform (2NF) wird erreicht, indem man die 1NF erfüllt und sicherstellt, dass alle Nicht-Schlüsselattribute voll funktional abhängig vom Primärschlüssel sind.

Im obigen Beispiel hängt der Name des Studenten direkt von der Matrikelnummer ab, nicht vom Kurs. Daher teilen wir die Tabelle auf, um diese Abhängigkeit aufzulösen.

Normalisierung in 2NF:

Tabelle 1: Studenten

MatrikelNr	Name
54321	Georg Egger
12345	Anna Bauer

Tabelle 2: Kursbelegungen

MatrikelNr	Kurs
54321	Datenbanken
54321	Mathematik
54321	Physik
12345	Informatik
12345	Datenbanken

5.3 Dritte Normalform (3NF)

Die dritte Normalform (3NF) wird erreicht, wenn die 2NF erfüllt ist und zusätzlich keine transitiven Abhängigkeiten zwischen Nicht-Schlüsselattributen bestehen.

Angenommen, wir erweitern unsere Tabelle um Professoren, die die Kurse leiten.

Erweiterte Tabelle (noch nicht in 3NF):

MatrikelNr	Kurs	Professor
54321	Datenbanken	Prof. Smith
54321	Mathematik	Prof. Doe
54321	Physik	Prof. Lee
12345	Informatik	Prof. Doe
12345	Datenbanken	Prof. Smith

Hier hängt der Professor vom Kurs ab, nicht von der Matrikelnummer des Studenten. Daher trennen wir diese Informationen.

Normalisierung in 3NF:

Tabelle 1: Studenten

MatrikelNr	Name
54321	Georg Egger
12345	Anna Bauer

Tabelle 2: Kurse

Kurs	Professor
Datenbanken	Prof. Smith
Mathematik	Prof. Doe
Physik	Prof. Lee
Informatik	Prof. Doe

Tabelle 3: Kursbelegungen

MatrikelNr	Kurs
54321	Datenbanken
54321	Mathematik
54321	Physik
12345	Informatik
12345	Datenbanken

Durch diese Schritte werden die Redundanzen minimiert und die Integrität der Datenbank erhöht. Jede Normalform baut auf der vorherigen auf und trägt dazu bei, die Datenbank effizienter und konsistenter zu gestalten.

6 Relationale Algebra

Die relationale Algebra ist ein theoretisches Modell für die Verarbeitung und Abfrage von Daten in einer relationalen Datenbank. Sie basiert auf Mengenoperationen und bietet eine Reihe von Operationen, die es ermöglichen, komplexe Datenabfragen und -manipulationen in einer formalen und präzisen Art und Weise durchzuführen. Jede Operation nimmt eine oder zwei Relationen als Eingabe und produziert eine neue Relation als Ausgabe.

Beispieldaten für die Beispiele:

1. **Studenten**

MatrikelNr	Name	Alter
1001	Max Müller	23
1002	Anna Bauer	21
1003	Tom Schmitz	22

2. **Kurse**

KursID	Kursname	Dozent
K01	Datenbanken	Prof. Meier
K02	Mathematik	Prof. Bauer
K03	Informatik	Prof. Weber

6.1 Projektion (Π)

Die Projektion ist eine Operation, die eine Untermenge der Attribute (Spalten) einer Relation auswählt und so eine neue Relation mit nur diesen Attributen erstellt. Duplikate werden entfernt, um die Eigenschaft einer Relation als Menge zu erhalten.

Beispiel:

$$\Pi\text{Name}(\text{Studenten})$$

Resultierende Tabelle:

Name
Max Müller
Anna Bauer
Tom Schmitz

6.2 Selektion (σ)

Die Selektion ist eine Operation, die eine Untermenge der Tupel (Zeilen) einer Relation basierend auf einer spezifischen Bedingung auswählt.

Beispiel:

$$\sigma\text{Alter} > 21(\text{Studenten})$$

Resultierende Tabelle:

MatrikelNr	Name	Alter
1001	Max Müller	23
1003	Tom Schmitz	22

MatrikelNr	Name	Alter

6.3 Join-Arten

6.3.1 Inner Join ($\bowtie$)

Kombiniert Tupel aus beiden Relationen, bei denen die Join-Bedingung wahr ist.

Beispiel:

Nehmen wir an, es gibt eine dritte Tabelle `Kursbelegungen`, die MatrikelNr und KursID verbindet:

MatrikelNr	KursID
1001	K01
1002	K02
1003	K03
1001	K03

Operation:

$$\text{Studenten} \bowtie_{\text{Studenten.MatrikelNr}=\text{Kursbelegungen.MatrikelNr}} \text{Kursbelegungen}$$

Resultierende Tabelle:

MatrikelNr	Name	Alter	KursID
1001	Max Müller	23	K01
1002	Anna Bauer	21	K02
1003	Tom Schmitz	22	K03
1001	Max Müller	23	K03

6.3.2 Left Outer Join ($\overset{\square}{\bowtie}$)

Enthält alle Tupel des Inner Joins sowie alle Tupel der linken Relation, für die es kein passendes Tupel in der rechten Relation gibt, ergänzt durch NULL-Werte.

6.3.3 Right Outer Join ($\rtimes\!\!\!\!\!\!\square$)

Wie Left Outer Join, aber für die rechte Relation.

6.3.4 Full Outer Join ($\square\!\!\!\!\!\!\bowtie\!\!\!\!\!\!\square$)

Kombiniert die Ergebnisse von Left und Right Outer Joins.

6.3.5 Natürlicher Join ($\bowtie$)

Ein spezieller Typ des Inner Joins, der automatisch Tupel auf Basis aller gleichnamigen Attribute in beiden Relationen verbindet.

6.3.6 Self-Join ($\ltimes_p$)

Ein Join einer Relation mit sich selbst, oft verwendet, um relationale Daten zu vergleichen.

6.3.7 Semi-Join ($\ltimes_p$)

Eine Operation, die die Tupel der ersten Relation auswählt, die in einer Beziehung zu einem Tupel der zweiten Relation stehen.

$$R \ltimes_p S = \pi_R(\ltimes_p S)$$

6.4 Kartesisches Produkt ($\times$)

Das kartesische Produkt von zwei Relationen ist eine Menge von Kombinationen jedes Tupels der ersten Relation mit jedem Tupel der zweiten Relation. Es wird selten direkt verwendet, aber ist die Grundlage für den Join.

6.5 Vereinigung ($\cup$)

Die Vereinigung zweier Relationen ist eine Menge, die alle Tupel enthält, die in mindestens einer der beiden Relationen vorkommen.

Beispiel: Nehmen wir an, es gibt zwei Tabellen mit Studentennamen:

Tabelle A:

Name
Max Müller
Anna Bauer

Tabelle B:

Name
Tom Schmitz
Lisa Neumann

Operation:

$$\text{Tabelle A} \cup \text{Tabelle B}$$

Resultierende Tabelle:

Name
Max Müller
Anna Bauer
Tom Schmitz
Lisa Neumann

6.6 Differenz ($\setminus$)

Die Differenz zweier Relationen ist eine Menge von Tupeln, die in der ersten Relation, aber nicht in der zweiten vorkommen.

Beispiel: Nehmen wir an, es gibt zwei Tabellen mit Studentennamen:

Tabelle A:

Name
Max Müller
Anna Bauer
Alex Benz
Tom Schmitz

Tabelle B:

Name
Tom Schmitz
Lisa Neumann

Tabelle A \ Tabelle B

Resultierende Tabelle:

Name
Max Müller
Anna Bauer
Alex Benz

6.7 Schnittmenge (∩)

Die Schnittmenge zweier Relationen ist eine Menge von Tupeln, die sowohl in der ersten als auch in der zweiten Relation vorkommen.

Beispiel:

Relation1 ∩ Relation2

Dieser Ausdruck wählt alle Tupel aus, die sowohl in Relation1 als auch in Relation2 vorkommen.

6.8 Umbenennung (ρ)

Die Umbenennung ist eine Hilfsoperation, die verwendet wird, um Relationen oder Attribute in einer Abfrage umzubenennen.

Beispiel:

$$\rho(\text{MatrikelNummer} \rightarrow \text{ID}, \text{Name} \rightarrow \text{StudentName})(\text{Studenten})$$

Resultierende Tabelle:

ID	StudentName
1001	Max Müller
1002	Anna Bauer
1003	Tom Schmitz

7 Anfrageoptimierung

Anfrageoptimierung ist die Überführung eines Ausdrucks in einen äquivalenten möglichst effizienten Ausdruck.

Generell gilt für Optimierung:

- **Frühestmögliche Selektion (σ)**: Selektionen reduzieren die Menge der Daten, die weiterverarbeitet werden müssen. Daher ist es i.d.R. günstiger, Selektionen so früh wie möglich durchzuführen.
- **Join ($\bowtie$) statt Kreuzprodukt ($\times$)**: Meistens sollen nur Zeilen kombiniert werden, die in einer Beziehungen zueinander stehen. Ein Join macht genau dies während ein Kreuzprodukt jede Zeile verbindet.
- **Frühestmögliche Projektion (π)**: Projektionsoperationen reduzieren die Anzahl an Spalten und reduzieren daher ebenfalls die Menge der Daten.

7.1 Kardinalitätsschätzung

Die Kardinalitätsschätzung dient der Kosteneinschätzung und errechnet / schätzt die Anzahl Tupes im Ergebnis einer Operation.

7.1.1 Selektion

Bei einer Selektion gibt es einen Selektivitätsfaktor $sf_p \in [0, 1]$, welcher bestimmt, wie viele Zeilen nach der Selektion übrig bleiben:

$$|\sigma_p R| = sf_p \cdot |R|.$$

Bei unabhängigen Selektionen gilt:

$$sf_{p \wedge q} = sf_p \cdot sf_q.$$

7.1.2 Kartesisches Produkt

$$|R \times S| = |R| \cdot |S|$$

7.1.3 Join

Im Allgemeinen:

$$0 \leq |R \bowtie_{R.a=S.a} S| \leq |R| \cdot |S|.$$

Wenn R.a Fremdschlüssel auf S.a ist, dann

$$|R \bowtie_{R.a=S.a} S| = |R|$$

8 SQL

SQL (Structured Query Language) ist eine standardisierte Sprache für die Definition, Manipulation, Abfrage und Steuerung von relationalen Datenbanken. Die wichtigsten Elemente von SQL umfassen:

Bekannte DBMS basierend auf SQL sind:

1. Oracle
2. MySQL
3. Microsoft SQL Server
4. PostreSQL
5. MongoDB
6. Redis
7. Elasticsearch

Gute Tutorials und Exercises gibt es auf w3schools

8.1 DDL (Data Definition Language)

- **CREATE**: Erstellt neue Tabellen oder Datenbankobjekte (Groß-Kleinschreibung egal). Beispiel: `CREATE TABLE Studenten (MatrikelNr INT NOT NULL, Name VARCHAR (100)DEFAULT "Not Set", email VARCHAR(100)CHECK (email LIKE '%@%' )UNIQUE);`
- **ALTER**: Modifiziert die Struktur bestehender Tabellen. Beispiel: `ALTER TABLE Studenten ADD Geburtsdatum DATE;`
- **DROP**: Löscht Tabellen oder andere Datenbankobjekte. Beispiel: `DROP TABLE Studenten;`

Spaltenoptionen:

- Primary Key: Primärschlüssel
- Not Null: Verbot von NULL-Werte
- Check: Bedingung für aktzeptierte Werte
- Unique: Eindeutigkeit
- Default: Standardwert
- References: Fremdschlüsselreferenz
- Auto_Increment

Zeilenoptionen:

- On Delete: Cascade (alle referenzierte Zeilen ebenfalls löschen), Set Null, Set Default, Restrict, No Action
- On Update: Analog zu on delete

Es kann auch eine neue Tabelle mit der gleichen Struktur wie eine andere Tabelle erstellt werden. Diese Tabelle kann auch direkt mit Daten aus dieser anderen Tabelle gefüllt werden: `CREATE TABLE schrottkarren AS SELECT * FROM pkw WHERE leistung < 60;`.

8.2 DML (Data Manipulation Language)

- **INSERT**: Fügt neue Datensätze in eine Tabelle ein. Beispiel: `INSERT INTO Studenten ( MatrikelNr, Name)VALUES (12345, 'Max Müller');`
- **UPDATE**: Aktualisiert bestehende Datensätze in einer Tabelle. Beispiel: `UPDATE Studenten SET Name = 'Max Mustermann'WHERE MatrikelNr = 12345;`
- **DELETE**: Entfernt Datensätze aus einer Tabelle. Beispiel: `DELETE FROM Studenten WHERE MatrikelNr = 12345;`

8.3 DQL (Data Query Language)

- **SELECT**: Wird verwendet, um Daten aus einer oder mehreren Tabellen abzufragen. Beispiel: `SELECT * FROM Studenten;`
  - Mit `JOIN`: Verknüpft zwei oder mehr Tabellen. Beispiel: `SELECT Studenten.Name, Kurse.Kursname FROM Studenten JOIN Kursbelegungen ON Studenten.MatrikelNr = Kursbelegungen.MatrikelNr JOIN Kurse ON Kursbelegungen.KursID = Kurse.KursID;` Ein `JOIN` kann `LEFT / RIGHT / FULL [OUTER] JOIN` sein (siehe w3schools). Die Spalte nach der welcher der Join gemacht werden soll kann auch mit `USING` gesetzt werden: `FROM produkte P JOIN bewertungen B USING(produktnr);`.
  - **Aggregatfunktionen**: Spezielle Funktionen, die auf eine Menge von Werten angewendet werden und ein einzelnes Ergebnis zurückliefern, wie `COUNT`, `SUM`, `AVG`, `MIN`, `MAX` und `GROUP BY`. Beispiel: `SELECT AVG(Alter)FROM Studenten;`
  - **Sub-Anfragen**: SQL-Abfragen innerhalb einer anderen SQL-Abfrage, die komplexe Abfragen ermöglichen. Beispiel: `SELECT Name FROM Studenten WHERE Alter > ( SELECT AVG(Alter)FROM Studenten);`
  - **Mengenoperationen**: Kombinieren die Ergebnisse von zwei oder mehr SELECT-Anweisungen, wie `UNION`, `INTERSECT`, und `EXCEPT`. Beispiel: `SELECT Name FROM Studenten_A UNION SELECT Name FROM Studenten_B;`

8.3.1 Views (Sichten)

8.3.1.1 Erstellung und Nutzung von Views Views, auch als Sichten bekannt, sind virtuelle Tabellen, die das Ergebnis einer Abfrage repräsentieren. Sie ermöglichen es, komplexe SQL-Abfragen zu vereinfachen und bieten eine abstrahierte Ansicht der Daten.

- **CREATE VIEW**: Zum Erstellen einer View. Beispiel: `CREATE VIEW StudierendeInformatik AS SELECT Name, MatrikelNr FROM Studenten WHERE Studiengang = 'Informatik';`
- **SELECT aus Views**: Views werden genauso abgefragt wie normale Tabellen. Beispiel: `SELECT * FROM StudierendeInformatik;`
- **UPDATE, INSERT und DELETE**: In manchen SQL-Systemen können Views auch für Update-, Insert- und Delete-Operationen genutzt werden, dies hängt jedoch von der View-Definition und den Datenbankbeschränkungen ab.

8.3.1.2 Check Optionen Die Check-Option in Views stellt sicher, dass alle Datenmanipulationen, die durch die View durchgeführt werden, die Bedingungen der View-Definition erfüllen.

- **WITH CHECK OPTION**: Diese Klausel gewährleistet, dass alle über die View eingefügten oder modifizierten Daten den Kriterien der View-Abfrage entsprechen. Beispiel: `CREATE VIEW StudierendeInformatik AS SELECT Name, MatrikelNr FROM Studenten WHERE Studiengang = 'Informatik'WITH CHECK OPTION;`

8.3.1.3 Materialisierte Sichten

Materialisierte Sichten sind physisch gespeicherte Views, die die Ergebnisse einer Abfrage speichern. Dies verbessert die Leistung bei häufig abgefragten oder komplexen Abfragen.

- **CREATE MATERIALIZED VIEW**: Erstellt eine materialisierte Sicht. Beispiel: `CREATE MATERIALIZED VIEW StudierendeInfoCache AS SELECT Name, MatrikelNr FROM Studenten WHERE Studiengang = 'Informatik';`
- **REFRESH**: Aktualisiert die Daten in einer materialisierten Sicht. Beispiel: `REFRESH MATERIALIZED VIEW StudierendeInfoCache;`

Materialisierte Sichten eignen sich besonders für Daten, die sich nicht häufig ändern, aber oft abgefragt werden. Sie bieten eine schnelle Abfrageantwort, da die Daten bereits in verarbeiteter Form vorliegen.

8.4 DCL (Data Control Language)

- **USER**: User können mit `CREATE USER` erstellt werden: `CREATE USER doktor_x WITH PASSWORD 'geheimespasswort1';`
- **GRANT**: Weist Benutzerrechte zu. Beispiel: `GRANT SELECT, INSERT, UPDATE ON Studenten TO Benutzer;`
- **REVOKE**: Entzieht Benutzerrechte. Beispiel: `REVOKE SELECT ON Studenten FROM Benutzer;`

Grants können auch zu einer Menge von User hinzugefügt werden: `GRANT SELECT ON ALL TABLES IN SCHEMA klinik TO big_boss;` oder `GRANT SELECT ON klinik.raeume TO public;`

8.5 TCL (Transaction Control Language)

- **COMMIT**: Bestätigt die Änderungen, die innerhalb einer Transaktion gemacht wurden.
- **ROLLBACK**: Macht Änderungen innerhalb einer Transaktion rückgängig.

8.5.1 ACID-Transaktionen in SQL

8.5.1.1 Atomicity (Atomarität)

- **Atomicity** bedeutet, dass eine Transaktion entweder vollständig oder gar nicht ausgeführt wird.
- SQL ermöglicht dies durch `BEGIN TRANSACTION` und `END TRANSACTION` (oder `COMMIT` und `ROLLBACK`).
- Beispiel: `BEGIN TRANSACTION; INSERT INTO Studenten (Name)VALUES ('Max'); UPDATE Kurse SET AnzahlStudenten = AnzahlStudenten + 1 WHERE KursID = 101; COMMIT;` Wenn ein Fehler auftritt, wird mit `ROLLBACK` die ganze Transaktion rückgängig gemacht.

SQL speichert Transaktionen erst im RAM, um die Perfomrnace zu verbessern. Nach einer gewissen Zeit werden die Transaktionen geflusht auf die Festplatte. Um sicherzustellen, dass bei einem Ausfall keine Daten verloren gehen wird jeder Commit in einem Transaktions-Log abgespeichert und kann im Fall eines Crashes wieder abgespielt werden, um auf den aktuellen Stand zu kommen.

8.5.1.2 Consistency (Konsistenz)

- **Consistency** stellt sicher, dass eine Transaktion die Datenbank von einem konsistenten Zustand in einen anderen überführt.
- Datenbankconstraints wie `PRIMARY KEY`, `FOREIGN KEY`, `CHECK`, und `UNIQUE` stellen sicher, dass die Konsistenz der Datenbank erhalten bleibt.

8.5.1.3 Isolation (Isolierung)

- **Isolation** sorgt dafür, dass Transaktionen, die gleichzeitig ausgeführt werden, sich nicht gegenseitig beeinflussen.
- SQL bietet verschiedene Isolationslevel, die mit `SET TRANSACTION ISOLATION LEVEL` gesetzt werden können, wie `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ`, und `SERIALIZABLE`.

8.5.1.4 Durability (Dauerhaftigkeit)

- **Durability** gewährleistet, dass einmal erfolgreich abgeschlossene Transaktionen auch bei einem Systemausfall erhalten bleiben.
- Dies wird durch Transaktionsprotokolle und regelmäßige Backups erreicht.

8.5.1.5 Implementierung in SQL

- **BEGIN TRANSACTION** startet eine neue Transaktion.
- **COMMIT** bestätigt die Änderungen einer Transaktion.
- **ROLLBACK** macht alle Änderungen einer Transaktion rückgängig, wenn ein Fehler auftritt.

- **SAVEPOINT** erstellt einen Wiederherstellungspunkt innerhalb einer Transaktion.
- **LOCK TABLES** und **UNLOCK TABLES** können verwendet werden, um manuelle Sperren auf Tabellen zu setzen und aufzuheben.

8.5.1.6 Beispiel einer ACID-Transaktion

ACID-Transaktion mit SQL:

```sql
BEGIN TRANSACTION;
INSERT INTO Studenten (MatrikelNr, Name) VALUES (12345, 'Max Mustermann
    ');
UPDATE Kurse SET AnzahlStudenten = AnzahlStudenten + 1 WHERE KursID =
    101;
IF @@ERROR <> 0
    ROLLBACK TRANSACTION;
ELSE
    COMMIT TRANSACTION;
```

8.6 Serialisierbarkeit und Sperrverfahren in SQL

8.6.1 Serialisierbarkeit

Serialisierbarkeit ist ein Konzept in der Datenbankverwaltung, das sicherstellt, dass die Ergebnisse von Transaktionen, die gleichzeitig ausgeführt werden, dieselben sind, als würden sie nacheinander ausgeführt. Dies ist das höchste Isolationslevel in SQL-Transaktionen.

8.6.1.1 Implementierung der Serialisierbarkeit

- **SET TRANSACTION ISOLATION LEVEL SERIALIZABLE**: Dieser Befehl setzt das Isolationslevel einer Transaktion auf serialisierbar.
- Beispiel: `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; BEGIN TRANSACTION ; ... COMMIT;`

8.6.2 SX-Sperrverfahren (Shared-Exclusive-Locking)

Das SX-Sperrverfahren ist eine Methode zur Verwaltung von Datenbankzugriffen, bei der es zwei Arten von Sperren gibt: Shared (S) und Exclusive (X).

8.6.2.1 Shared Lock (S)

- Ein Shared Lock wird gesetzt, wenn eine Transaktion Daten liest.
- Mehrere Transaktionen können gleichzeitig Shared Locks auf dieselben Daten halten.

- Beispiel: `SELECT * FROM Studenten WITH (HOLDLOCK);`

8.6.2.2 Exclusive Lock (X)

- Ein Exclusive Lock wird gesetzt, wenn eine Transaktion Daten ändert (INSERT, UPDATE, DELETE).
- Keine andere Transaktion kann gleichzeitig einen Shared oder Exclusive Lock auf diese Daten halten.
- Beispiel: `UPDATE Studenten SET Name = 'Max Mustermann'WHERE MatrikelNr = 12345;`

8.6.3 Deadlock-Vermeidung

Ein Deadlock tritt auf, wenn zwei oder mehr Transaktionen auf Sperren warten, die von den anderen Transaktionen gehalten werden, wodurch ein Kreislauf der gegenseitigen Blockierung entsteht.

8.6.3.1 Deadlock-Erkennung und -Behebung

- **Deadlock-Erkennung**: Moderne Datenbanksysteme haben integrierte Mechanismen zur Erkennung von Deadlocks.
- **Deadlock-Behebung**: Wenn ein Deadlock erkannt wird, wählt das System eine der Transaktionen aus und macht sie rückgängig (ROLLBACK), um den Deadlock zu lösen.

8.6.3.2 Beispiele und Strategien zur Vermeidung von Deadlocks

- **Timeouts verwenden**: Wenn eine Transaktion nicht innerhalb eines bestimmten Zeitraums abgeschlossen werden kann, wird sie zurückgesetzt.

8.6.4 Beispiel für Serialisierbarkeit und Sperrverfahren

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;

SELECT * FROM Kurse WITH (ROWLOCK); -- Shared Lock
UPDATE Kurse SET AnzahlStudenten = AnzahlStudenten + 1 WHERE KursID =
    101; -- Exclusive Lock

IF @@ERROR <> 0
    ROLLBACK TRANSACTION;
ELSE
    COMMIT TRANSACTION;
```